PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1984,
edited by ICOT. © ICOT, 1984

QUERY PROCESSING FLOW ON RDBM DELTA'S FUNCTIONALLY-DISTRIBUTED ARCHITECTURE

Shigeki Shibayama, Takeo Kakuta, Nobuyoshi Miyazaki, Haruo Yokota, and Kunio Murakami

ICOT Research Center
Institute for New Generation Computer Technology
Tokyo, Japan

ABSTRACT

This paper presents details on the implementation of relational database machine Delta, being developed at the Institute for New Generation Computer Technology (ICOT). Using an example, we focus on how a transaction consisting of Delta access commands are received, analyzed, managed and executed. This paper is mainly concerned with the software configuration and functional capabilities distributed among the processors that constitute Delta. A detailed description of the relational database engine (RDBE) will be given in another paper (Sakai 84). An approach to the use of Delta as a research tool is also briefly described.

1 BACKGROUND AND INTRODUCTION

In Japan's Fifth Generation Computer System (FGCS) project, a relational database machine is being developed. The relational database machine (**Delta**) will be finished at the end of the project's initial 3-year stage. It will then be used as a back-end storage to sequential inference machine (**SIM**) users in the intermediate 4-year stage. An FGCS prototype is shown in Figure 1. Delta receives concurrent queries issued from the SIM via a local area network. Delta will be used not only as a back-end storage for the network, but also as a research tool for further investigations of knowledge base machines. An experimental system in which a SIM and Delta are tightly-coupled will be used for this purpose.

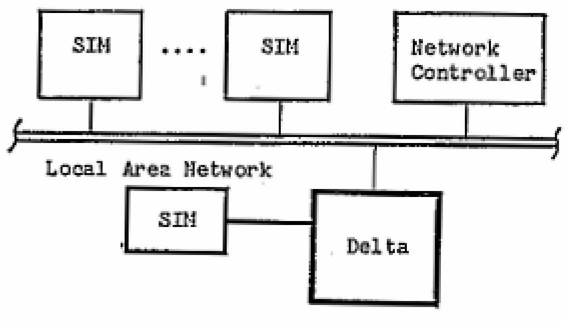

Figure 1. An FGCS Prototype

Delta is basically a relational database system featuring a hardware relational-type database processor and a large capacity semiconductor disk (Shibayama 84). It is a functionally-distributed multi-processor system provided with most of the conventional database management facilities, for example, multi-user support and recovery functions.

In this paper, the basic architectural concept is described in Chapter 2 and the hardware configuration is described in Chapter 3. Chapter 4 describes in detail the software configuration and how a transaction is decomposed, dispatched and executed across the subsystems. Research plans for developing a knowledge base machine are briefly discussed in Chapter 5.

2 ARCHITECTURE

A conceptual diagram of Delta's architecture is shown in Figure 2. Delta's relational database machine design uses a functionally-distributed architecture. By separating functions required for constituting a database management system, efficient hardware and software can be implemented for each of the functional subsystems. We divided the database functions into five categories. Those are: (1) host interfacing, (2) query analysis and hardware

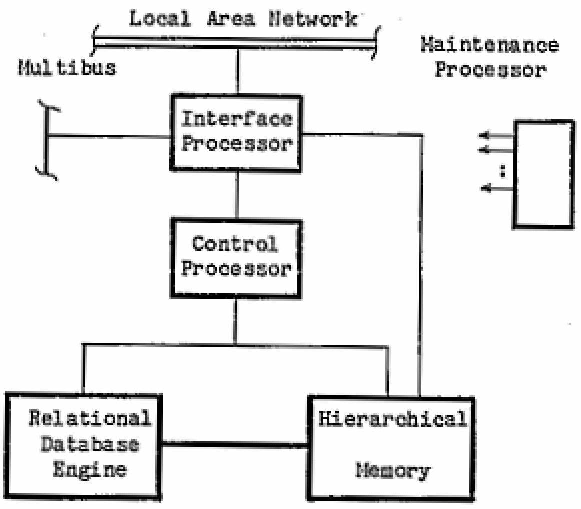

Figure 2. Delta Architecture

resource management in the subsystem level, (3) relational database processing, (4) database storage management, and (5) system supervising. We provided an Interface Processor (**IP**), a Control Processor (**CP**), Relational Database Engines (**RDBE**), a Hierarchical Memory (**HM**) and a Maintenance Processor (**MP**), respectively, to perform the above distributed functions. The implementation of each subsystem is described in the following chapter. The summary of functions distributed to the subsystems are as follows:

(1) **Interface Processor (IP)**

- **Local area network interface (physical and logical)**

  This interface provides a loosely-coupled connection.
- **A tightly-coupled interface**

  This is an experimental interface with a SIM.
- **Concurrent command-sequence management**

  Delta provides a multi-user environment. The IP acts as a front-end for Delta and resolves command-sequence contentions from multiple SIMs.
- **Tuple transfer**

  IP has a high-bandwidth transfer path to HM, which is needed to perform this function.

(2) **Control Processor (CP)**

- **Transaction management**

  The transaction management function is responsible for tracking the status of each transaction and for database resource management.
- **Command-tree management**

  Delta command (command-tree) analysis, subcommand generation, and subcommand execution control are performed by this manager.
- **Dictionary/directory management**

  Delta's dictionary is a set of meta-relations to which users can refer. The directory is an internal data structure that the CP software refers to. Consistency must be maintained between them at transaction commit or abort times.
- **Recovery management**

(3) **Relational Database Engine (RDBE)**

- **Relational database processing**

  To execute relational algebra operations and set operations fast, this section requires special-purpose hardware and software. We observed that the hardware two-way merge-sorting could be used to carry out fast relational database operations (Sakai 84). The simultaneous execution of data transfer and relational database operation is useful when the data amount is fairly large. To enjoy the fast engine hardware, a bi-directional high-bandwidth path to the storage portion is necessary. This requirement led to the incorporation of a large semiconductor memory with high-speed channels into the Delta HM subsystem.

(4) **Hierarchical Memory (HM)**

- **Cache and moving-head-disk management**

  As Delta must accommodate current technologies, the omission of moving-head-disks for the storage device is unthinkable. The HM is responsible for preparing and receiving source and result relations (data) at a high rate of transfer with RDBE, using a large semiconductor memory.
- **Data recovery**

  Another important role assigned to the HM is the data recovery. This must be performed, of course, in cooperation with the CP and MP recovery management sections.

(5) **Maintenance Processor (MP)**

- **System supervising**

  The Maintenance Processor (MP) is responsible for system supervision. As Delta is a large computer complex, a global system maintenance section is needed. The following lists show these tasks in a little more detail:
- **Start-up and shut-down of each subsystem**
- **Subsystem status management**
- **Operator command execution**
- **Delta system state management**
- **Database loading and saving**
- **Statistical data collection**

3 HARDWARE CONFIGURATION

The hardware configuration of the Delta system is shown in Figure 3. We focused on the implementation of RDBE in the hardware system development. RDBE's hardware organization is shown in Figure 3. A two-way merge-sorter and a merger are responsible for most relational algebra operations and set operations. The merge-sorter is a one-dimensional special-purpose processor array for pipelined merge-sorting (Todd 78). The merge-sorter can sort N items in $(2 \times N + \log_2 N)$ time units, where a time unit is the time required to transfer one item. A set of four RDBE's are installed for parallel transaction execution. Each RDBE can work independently to execute separate queries or they can all work cooperatively to execute a single relational database operation. The details of RDBE's load distribution have not yet been determined. An RDBE has a general-purpose mini-computer CPU with 512KB of main storage for controlling the hardware section called the Engine Core and performing the RDBE commands that the hardware section does not support.

The IP, CP and MP make use of the same general-purpose mini-computers as RDBE. The sizes of IP, CP and MP main storage are 1MB, 1MB and 512KB, respectively. Each processor is provided with a local disk storage for the operating system use. The CP disk is a semiconductor disk storage. The IP and MP are provided with Winchester-type unremovable disk drives. CP's control program temporarily stores the directory and other tables in the semiconductor disk storage. This technique has been adopted for rapid access to information swapped out in the secondary storage.

The HM is implemented using a larger general-purpose CPU as its controller. The main storage of the CPU is used for both program storage and as a disk cache. The size of main storage is 128MB. Main storage will be made non-volatile (at least from the software's point of view) by an emergency power supply to avoid disk accesses invoked by a write-through storage management and to facilitate database recovery. The moving-head-disks have a capacity of about 600MB per volume. Each pair of volumes is packaged in a housing along with its own head assemblies and share a common motor drive (spindle). Thirty-two such volumes are attached to the I/O channels; thus, a total storage capacity of about 20GB is provided.

The CP, IP, RDBE and MP form a group of processors called the RDBM supervisory processing subsystem (RSP for short). All the RSP constituents use the same CPU. The HM is a different subsystem and has its own CPU. Communication between the RSP constituents and HM is performed via a standard channel interface. The channel transfers data at speeds up to 3MB/second. An RDBE is provided with two independent channels to the HM for dedicated I/O use. Each of the other RSP constituents are provided with a single channel for bi-directional communication. Internal communication in RSP constituents are performed via IEEE488 buses. The transfer speed of the bus is about 150KB/second. As shown in Figure 3, the buses are independently installed between the RSP constituents.

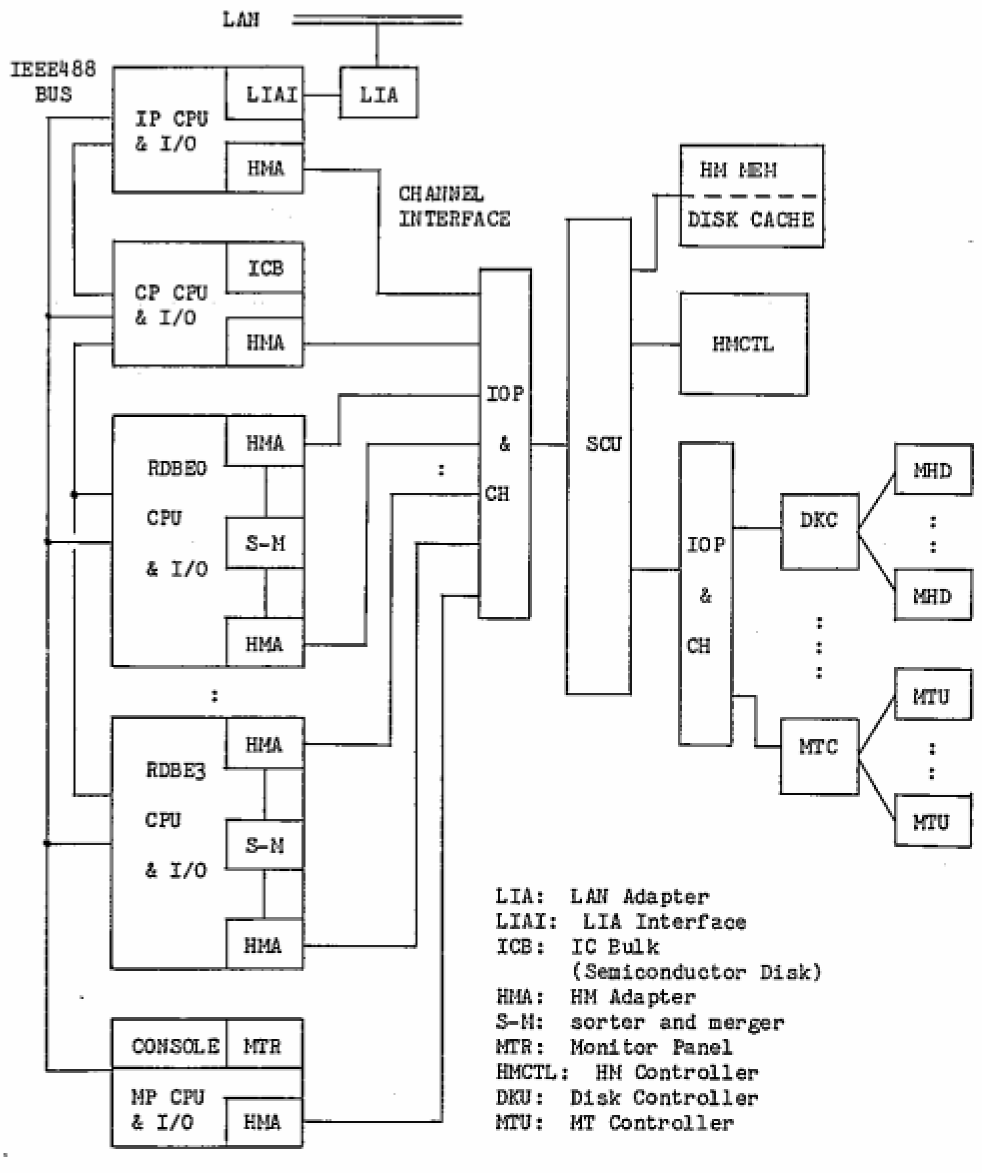

Figure 3. Delta Hardware Configuration

Table 1. Dictionary and Directory

name	definer updater	reference	external interface	implementation	HM interface
dictionary	host* Delta	host	relation	permanent relation	permanent relation
directory	Delta	Delta	not defined	linked data	page

*: Only qualified attributes can be updated by hosts.

4 FUNCTION DISTRIBUTION OF DELTA

4.1 Software Configuration

Some pieces of software are common to IP, CP, and MP. Principal among these is the operating system. The operating system is a modified version of an existing one and supports multi-tasking. The IEEE488 bus driver is used for inter-subsystem communications, except for those with the HM. In the IP, the IEEE488 driver is also used as the LIA (LAN Adapter) physical level interface. The RDBE has the same CPU board as the controller. However, its software does not have to perform concurrent operations, so an operating system is not used.

(1) IP

The IP is responsible for communicating with SIMs in the LAN environment. The IP is physically connected to the LIA by an IEEE488 bus. The physical level LIA interface in IP is performed by the **LIA communication task**. The logical level LIA interface is performed by an **IP control task**. The IP has multiple **command-tree buffers** for receiving multiple command-trees sent to it concurrently. The IP **command-tree management task** queues the command-trees sent from a host in order of their arrival in the command-tree buffer. The **command-tree management task** then forwards the command-trees in the buffer to the CP using a **CP communication task**. As the physical connection between the IP and the CP is also an IEEE488 bus, this task uses an IEEE488 bus driver, identical to the LIA physical level interface.

Usually, the Delta command-trees in a transaction are taken out serially from the command-tree buffer. However, asynchronous-type commands such as **sense-status** and **abort-processing** are processed as soon as possible when they arrive at IP and are recognized.

(2) CP

The CP is responsible for controlling transactions, managing database and Delta resources, command analysis, issuing subcommands, dictionary/directory (D/D) management, rollback and statistical data collection.

When the CP is triggered by the transfer of the **start-transaction** command from the **IP command-tree management task** via the **CP communication task**, a **command-tree processing task** is created. A **command-tree processing task** corresponds to a transaction. The task is killed upon termination of the transaction. An operating system facility (**transaction supervisor task**) is notified of the states of the command-tree processing tasks.

The **command-tree processing task** gets command-trees one after another. It analyzes the command-tree, and if necessary, locks permanent relations at execution time. It then generates a sequence of subcommands for RDBE and HM by referring to the directory for schema checking. After all the commands are translated into their corresponding RDBE and HM subcommand sequences, a **command-tree processing task** forwards these sequences to the **RDBE and the HM communication task** for distribution to the subsystems. At execution time, by checking the responses from RDBE and HM, the **command-tree processing task** determines the parameters to appear in the generated subcommand sequence. For example, if the resultant temporary relation of a command is found to contain no tuples at all, the subsequent subcommand sequence is skipped.

The **dictionary/directory (D/D) management task** is responsible for maintaining the directory for internal use and the dictionary as a user reference. The differences between the dictionary and the directory are summarized in Table 1. Dictionary relations consist of two meta-relations, **"Relations"** and **"Attributes"**. Essentially, the dictionary and directory hold the same schema information. However, dictionary attributes unnecessary for the schema checking are not included in the directory. The reasons we separated the dictionary and directory are: (1) the efficiency in look-up, and (2) the concurrent access control. Usually, relations are locked to manage concurrency control. If the dictionary is also locked during a look-up, locking conflicts will be frequent. Instead, we use the directory for finer concurrency control. The HM supports the directory storage (directory area) separate from the disk cache or program area. The CP semiconductor disk cache stores the directory for repeated use. If necessary, the CP obtains directory pages from the HM by directory-access subcommands.

As Delta is used in a network environment, there is a problem with the consistency of dictionaries. Each Delta control program in hosts will contain its own dictionary. A Delta control program has no way of knowing when another control program's dictionary has been updated. To resolve this problem, an attribute (**Redefined-at**) is provided for the **"Relations"** relation to keep the dictionary current. In the "**Redefined-at**" attribute, the time of the last redefinition is recorded. Every permanent relation access must be associated with this value. If it does not match the current value, Delta informs the host that the dictionary is obsolete. The host must read the dictionary relations in order to continue.

The **concurrency control task** is a subtask of the **transaction supervisory task**. This task manages concurrency among transactions. Concurrency is controlled by locking the resources (relations). We selected the two-phase locking method. Permanent relations to be locked during a transaction can be explicitly locked using a **start-transaction** command, or they can be automatically locked before command-tree execution. All the relations are unlocked at the end of a transaction whether it is a normal or an abnormal termination.

(3) **RDBE**

The RDBE does not have an operating system. Its entire program works as a single task. The software structure of RDBE consists of three layers: the uppermost layer controls subcommand execution; the middle layer controls the I/O traffic; and the lowest layer controls interrupt handling.

The **subcommand execution layer** receives subcommands from the CP and MP, analyzes and executes them and returns responses to their senders. Basically, **subcommand execution layer** repeats this cycle. It calls the **I/O traffic control layer** as libraries for handling I/O devices. This layer also performs extended operations that the hardware core section does not support. The operations are listed below:

- Filtering of result data by a complex criteria

  The hardware core section can perform operations only with a simple criterion such as a range search. The conjunction of one-term criteria, for example, is performed within the CPU.
- Arithmetic operations on attribute fields and result value assignment
- Aggregate operations on grouped streams

These operations are specified as parameters of RDBE subcommands.

The **I/O traffic control layer** provides library routines for the **subcommand execution layer** and interrupt handling routines for the **interrupt handling layer**. Recoverable I/O errors are for the most part corrected here.

The **interrupt handling layer** performs the following tasks:

- It establishes an environment in which RDBE control program runs at start-up time.
- It calls an interrupt handling routine provided by the **I/O traffic control layer** by checking the I/O device status when an interrupt occurs.
- It handles other interrupts such as CPU internal interrupts.
- It provides libraries for manipulating special machine-dependent instructions, such as PSW modifications.

(4) **HM**

The HM software configuration is as follows:

- **Operating system**
- **HM control program**
  - **Control module**

    This module consists of an **HM task control submodule**, which manages HM multi-tasking, and an **RSP interface submodule** responsible for RSP interfacing.
  - **Subcommand processing module**

    This module processes various HM subcommands issued from other subsystems. The subcommands are classified by processing type.
  - **Common function module**

    This module is a collection of functions used among other sections. Cluster management, log management, recovery management, memory management and disk space management are its principal tasks.
  - **Initialisation and termination module**
- **Support programs**
- **Test programs**

The HM control program performs the main database operations specified in the form of subcommands. Each module in the program is further divided into submodules corresponding to logical units of HM internal operations. These submodules run under the **HM task control submodule**. Subcommands from RSP are collectively managed by the **RSP interface submodule**. A **subcommand process submodule** is organized to correspond to a set of RSP subcommands. For example, the **attribute definition/operation submodule** is activated to a set of attribute definition-type subcommand sequences. The HM is usually a passive subsystem. It is activated upon receipt of a subcommand.

4.2 A Sample Transaction Processing Flow

As described in the architecture chapter, Delta satisfies a set of database machine requirements by adopting a functionally-distributed configuration. By showing how a sample transaction is received, analyzed and executed, we will illustrate the functional capabilities distributed among Delta subsystems.

A **transaction** is a group of command-trees beginning with a **start-transaction** command and ending with an **end-transaction** command. When the database update is specified, a transaction is the unit of update: any unsuccessful transaction is rolled back to its state previous to the start-transaction command; a successful transaction is fully updated. In a read-only sequence of Delta commands, a transaction defines a scope in which intermediate relations (typically a result of a command-tree) are stored and used across command-trees. Once a **commit-transaction** command is received, only **input/output type commands** are accepted until the end-transaction command is received.

We will consider the following sample transaction. The permanent relations used are **company(company_**

name, location) and **icot(name, age, company, laboratory, group)**. The following query selects the names, laboratory affiliation and group name of the persons within ICOT who were formerly employed by a company in Yokohama. This query is written as follows in a SQL-like notation:

```
start-transaction
select name, laboratory, group
from icot
where company = next
select company_name
from company
where location = [yokohama]
end-transaction.
```

This query is translated into the Delta command sequence by a translator (in SIM software) as follows (the parameters are modified or abbreviated for readability):

(1.1) **Start-transaction**

define the beginning of a transaction scope

(2.1) **Selection(company,[2]=[yokohama],temp1)**

select from company relation in which the second ([2]) attribute equals yokohama, place resultant relation into a temporary relation temp1

(2.2) **Projection(temp1,[1],temp2)**

project the temp1 relation against the first ([1]) attribute into temp2 relation

```
[ subcommand group 1 ]
/* generation of tid1:company-location pair */
CP→HM: prepare-qualified-buffer(buf1);location=yokohama
CP→HM: prepare-buffer(buf2);output buffer
CP→RDBE: restrict;company-location=yokohama
RDBE→HM: start-stream-in(buf1);tid list is obtained
RDBE→HM: start-stream-out(buf2);buf2=tid1
[ subcommand group 2 ]
/* sorting of tid1 */
CP→HM: prepare-buffer(buf3);buffer for sorted tid
CP→RDBE: sort;sorting RDBE command
RDBE→HM: start-stream-in(buf2)
RDBE→HM: start-stream-out(buf3);buf3=tid1 (sorted)
[ subcommand group 3 ]
/* making tid1:company-name pair from tid1 */
CP→HM: prepare-qualified-tid-buffer(buf4)
CP→HM: prepare-buffer(buf5);output buffer
CP→RDBE: restrict;tid1 selection
RDBE→HM: start-stream-in(buf4);tid1:company-name
RDBE→HM: start-stream-in(buf3);tid1 (sorted)
RDBE→HM: start-stream-out(buf5);tid1:company-name
[ subcommand group 4 ]
/* semi-join of icot-company and company-name */
CP→HM: prepare-qualified-buffer(buf6);icot-company
CP→HM: prepare-buffer(buf7);tid triplets output buffer
CP→RDBE: join;company-name=icot-company
RDBE→HM: start-stream-in(buf5);company-name
RDBE→HM: start-stream-in(buf6);icot-company
RDBE→HM: start-stream-out(buf7);tid triplets
[ subcommand group 5 ]
/* icot tid sort */
CP→HM: prepare-buffer(buf8);output buffer
CP→RDBE: unique;icot tid extract and sort
RDBE→HM: start-stream-in(buf7);tid triplets buffer
RDBE→HM: start-stream-out(buf8);tid2 (sorted)
[ subcommand group 6 ]
/* icot-name attribute selection */
CP→HM: prepare-qualified-tid-buffer(buf9);icot-name
CP→HM: prepare-buffer(buf10)
CP→RDBE: restrict
RDBE→HM: start-stream-in(buf9);icot-name
RDBE→HM: start-stream-in(buf7);tid triplets
RDBE→HM: start-stream-out(buf10)
[ subcommand group 7 ]
/* icot-lab attribute selection */
CP→HM: prepare-qualified-tid-buffer(buf11)
CP→HM: prepare-buffer(buf12)
CP→RDBE: restrict
RDBE→HM: start-stream-in(buf11)
RDBE→HM: start-stream-in(buf7)
RDBE→HM: start-stream-out(buf12)
[ subcommand group 8 ]
/* icot-group attribute selection */
CP→HM: prepare-qualified-tid-buffer(buf13)
CP→HM: prepare-buffer(buf14)
CP→RDBE: restrict
RDBE→HM: start-stream-in(buf13)
RDBE→HM: start-stream-in(buf7)
RDBE→HM: start-stream-out(buf14)
[ subcommand group 9 ]
/* icot-name attribute sort by tid */
CP→HM: prepare-buffer(buf15)
CP→RDBE: sort
RDBE→HM: start-stream-in(buf10)
RDBE→HM: start-stream-out(buf15)
[ subcommand group 10 ]
/* icot-lab attribute sort by tid */
CP→HM: prepare-buffer(buf16)
CP→RDBE: sort
RDBE→HM: start-stream-in(buf12)
RDBE→HM: start-stream-out(buf16)
[ subcommand group 11 ]
/* icot-group attribute sort by tid */
CP→HM: prepare-buffer(buf17)
CP→RDBE: sort
RDBE→HM: start-stream-in(buf14)
RDBE→HM: start-stream-out(buf17)
[ subcommand group 12 ]
/* tuple reconstruction */
CP→HM: transpose-to-tuples
/* result tuple transfer */
IP→HM: start-packet-in;for get
IP→HM: start-packet-in;for get-next
```

The XX→YY: prefix denotes subcommand issuance from XX to YY. The instructions next to the prefix are subcommands. Semicolon begins a comment.

Figure 4. HM and RDBE subcommand sequence

(2.3) **Projection(icot,[1,3,4,5],temp3)**

project icot relation against the first, third, fourth and fifth attributes into temp3

(2.4) **Natural-join(temp3,temp2,[2]=[1],temp4)**

natural-join temp3 and temp2 with the second and the first attributes, respectively, put into temp4

(2.5) **Projection(temp4,[1,3,4],int1)**

project temp4 against the first, third and fourth attributes into an intermediate relation int1

(3.1) **Commit-transaction**

freeze the transaction; freezing means to inhibit further modification of the resultant intermediate relation in the case of read-only transactions

(4.1) **Get(int1)**

fetch the intermediate relation (int1) from the top tuple

(5.1) **Get-next(int1)**

fetch the intermediate relation subsequently

(6.1) **End-transaction**

conclude the transaction

This command-sequence consists of six command-trees; the first command number (before the decimal point) denotes a command-tree number, the second number (after the decimal point) denotes a command number within a command-tree. Each transaction-control command and input/output-type command forms their own command-tree. The six command-trees are packed with a chain identifier, command-tree identifiers and physical delimiters by SIM's translator software. This packing is called a command-tree chain. The command-tree chain is transferred as a unit in a sequence of LIA commands (LAN packets). The LAN interface adapter (LIA) is a LAN subsystem responsible for interfacing between SIM and Delta. The command-tree chain are then forwarded to the SIM network subsystem (NS), which deals with the local area network interfacing task.

NS uses predetermined network protocols to send the command-trees to Delta. The first thing that NS must do is to form a **communication-group** with Delta for each user database job. A user job consists of a set of serialized transactions.

The sample command sequence is translated into the RDBE and HM subcommand sequence shown in Figure 4.

Delta's internal storage schema is **attribute-based**. Every attribute is stored separately with a **tuple-identifier (tid)** and tag fields. Therefore, the subcommand sequence does not directly correspond to the Delta command sequence. With a tuple-based schema, the order of the relational database operations affects performance. For example, projections are performed where appropriate, to reduce unnecessary attribute handling. With an attribute-based schema, as the subcommands only work on attributes that appear explicitly in the operations, the order of projection commands is of less importance. Other optimizations, for example, selections before a join, are still effective using the attribute-based schema.

Subcommand group 1 (SG1 for short) in Figure 4 filters the tid fields of the **company-location** attribute, the value of which is "yokohama" in buf2. SG2 sort the tid fields in buf2 into buf3. SG3 joins the tid fields with the **company-name** attribute's tid field. The **company-name** attribute values are obtained in buf5. SG4 joins the **company-name** attribute values in buf5 with **icot-company** attribute values. This corresponds to the **natural-join** command. The content of buf5 is a set of triplets of tid's, that is, the tid of the **company** relation, the tid of the **icot** relation and a new tid for the relation generated by the join. Buf5 is a form of intermediate relation and is the result of the second command-tree, specified as "int1". If "int1" is used by another command, the following subcommand sequence is changed. In this example, the next command-tree is a **commit-transaction**, so the reconstruction process of tuples into output form (usual tuple-based form) takes place. SG5 extracts and sorts the tids of the **icot** relation from the triplet buffer for later tid-restriction, that is, selection of other attributes needed for the output relation. SGs 6, 7 and 8 select the corresponding attributes, **icot-name, icot-lab, icot-group**, respectively, for the selected **icot** tids. The output attributes are then sorted and reconstructed into tuple form by SGs through 12. The buffers used in the transaction are dynamically released within the transaction. The buffers for the output form relation are released when the **end-transaction** command is received.

5 DELTA IN A LOGIC PROGRAMMING RESEARCH ENVIRONMENT

Delta is used as a back-end database storage in the intermediate stage of the Fifth Generation Computer Systems project. A number of SIM machines are connected via the ICOT's local area network (Taguchi 84). Delta is accessed from a Delta interface program (called **RDBMS** for relational database machine management system) and from the SIM operating system.

There are several software layers through which user access Delta (Figure 5). The lowest layer is responsible for handling the **physical network protocols**. This layer corresponds to the IP **LIA communication task**. The next layer handles logical network protocol. Most of the network layer is supported by the SIM operating system's network subsystem (NS).

The **translator layer** will be supported by the **RDBMS**. During the translation process, RDBMS refers to the prefetched dictionary relations, generates and manages transactions, controls access rights for security and performs related operations.

We anticipate the following uses:

(1) Users handle logical Delta commands directly. This uses the **physical Delta command translation layer**.

(2) Users define (in programs) special predicates and write programs in the usual way. The RDBMS **relational algebra translation layer** is responsible for both the interpretation of user programs and the transformation of relational database queries into Delta-command form. This method is presented in (Yokota 84).

- Network layer (NS)
  - Physical network layer
  - Logical network layer
- Translator layer (RDBMS)
  - Physical Delta-command translation layer
  - Logical Delta-command translation layer
  - Delta command generation layer
- Transaction management (RDBMS)
  - Dictionary management
  - Access right (security) control
  - Resource management
  - Transaction generation/deletion
  - Command execution control

Figure 5. RDBMS Software Layers

(3) Users will have to write their own translation layer programs for special-purpose software application systems. The output of the program is passed to the **logical Delta interfacing layer**.

(4) Users need only use a high-level database query language, for example, SQL or QBE, which make use of the RDBMS functions.

In the local area network environment, however, the LAN communication overhead is too large for interactive database access. We proposed a method of combining a logic programming language with the relational database. This corresponds to the second usage in the list. In this method, Delta accesses are collected and issued in a batch, because of the access characteristics of the LAN.

We will also provide a more tightly-coupled SIM-to-Delta interface. We think that it is necessary to tight-couple the inference mechanism with the database access mechanism to make a knowledge base machine dealing with vast amounts of knowledge for performing inference.

The following are a few of the approaches for developing a knowledge base machine based on logic programming.

(1) Addition of a virtual memory system to the SIM architecture to store the entire knowledge base along with inference procedures

(2) Establishment of a tightly-coupled interface between the logic programming language and knowledge base

(3) Investigation of totally new architectures for manipulating knowledge and inference procedures

The first approach does not seem promising, because the internal data structure required to perform inference and that required to access a large amount of data are very different. However, for programmers this one-level storage treatment would be optimal. A similar method must be found to facilitate programmers' access to the knowledge base system. However, we do not consider this a good approach for implementation.

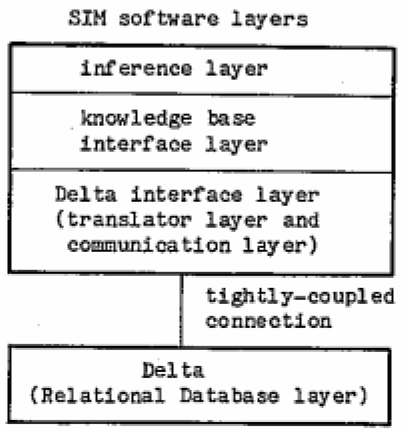
SIM software layers
inference layer
knowledge base interface layer
Delta interface layer (translator layer and communication layer)
tightly-coupled connection
Delta (Relational Database layer)

Figure 6. Software layers for a Knowledge Base Experimentation

The third approach seems to be premature in view of the current state of knowledge base research. To develop a new architecture, a clear direction and sound principle are required. Not until the research stages have been completed will such principles be subject to effective investigation.

We think that the second approach is most practical given the resources currently available. An interface between the logic programming language and the knowledge base section must be thoroughly investigated. The relational database concept is not fully appropriate for the basis for such an interface. We consider the **unit-clause** interface to be better suited for the knowledge base model in conjunction with a logic programming language (Yokota 83). This implies that a simple **unification** capability is a natural extension of the relational model and provides a basis for a knowledge base model.

To develop an experimental knowledge base system in keeping with these assumptions, a tightly-coupled connection between SIM and Delta is needed. As LANs typically cause some amount of transfer overhead, a faster and more responsive interface is necessary for the knowledge base machine. We decided to use the buffer memory in the SIM system as a **communication memory**. The I/O buffer memory is connected to the IEEE796 bus (Multibus) under an I/O controller. The path between Delta and SIM is established by adding an interface board to the SIM system for accessing the bus and connecting it to the IP. We will create a software system for SIM that closely-couples the logic programming language to the relational database. By providing a software layer hierarchically superior to RDBMS, we can simulate an experimental interface between the knowledge base and inference mechanism. Thus, we can vary the interface levels. We also plan to investigate the nature of appropriate interfaces for future knowledge base machines (Figure 6).

6 CONCLUSION

We have described the functionally-distributed architecture of Delta, its hardware and software configuration. We have presented a detailed processing flow by examining a sample transaction. The relationship of Delta to a logic programming environment and future research plans for a knowledge base machine are described. More accurate performance estimates based on measurements made on the actual machine, represent the next step in evaluation. Research on knowledge base mechanisms will be the focus of the last part of the project's initial stage.

ACKNOWLEDGMENTS

The authors wish to express their appreciation to the Toshiba and Hitachi researchers and engineers who assisted with the implementation and design for their efforts in this critical undertaking.

REFERENCES

Sakai, H., Iwata, K., Kamiya, S., Abe, M., Tanaka, A., Shibayama, S., Murakami, K. Design and Implementation of the Relational Database Engine. *Proc. 2nd FGCS Conference*, Nov., 1984.

Shibayama, S., Kakuta, T., Miyazaki, N., Yokota, H., Murakami, K. A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor. *ICOT Technical Report TR-055* and also in *New Generation Computing*, Vol.2, No.2, May, 1984.

Taguchi, A., Miyazaki, N., Yamamoto, A., Kitakami, H., Kaneko, K., Murakami, K. INI: Internal Network in ICOT and its Future. *Proc. 7th ICCC*, Oct., 1984.

Todd, S. Algorithm and Hardware for a Merge Sort Using Multiple Processors. *IBM Journal of Res. and Develop.*, 22, 1978.

Yokota, H., Kakuta, T., Miyazaki, N., Shibayama, S., Murakami, K. An Investigation for Building Knowledge Base Machines. *ICOT Technical Memorandum TM-0019*, 1983.

Yokota, H., Kunifuji, S., Kakuta, T., Miyazaki, N., Shibayama, S., Murakami, K. An Enhanced Inference Mechanism for Generating Relational Algebra Queries. *Proc. 3rd ACM SIGACT-SIGMOD symposium on Principles of Database Systems*, pp. 229 - 238, April, 1984.